Session: G07

DB2 9: For Developers Only!

Craig S. Mullins

NEON Enterprise Software, Inc.

NEON Enterprise Software, Inc.

IDUG
The Worldwide DB2 User Community

May 13, 2009 • 01:30 p.m. – 02:30 p.m.
Platform: z/OS

Authors

This presentation was prepared by:

Craig S. Mullins

Corporate Technologist

NEON Enterprise Software, Inc.
14100 Southwest Freeway, Suite 400
Sugar Land, TX 77478
Tel: 281.491.6366
Fax: 281.207.4973
E-mail: craig.mullins@neonesoft.com

NEON Enterprise Software, Inc.

Agenda

This presentation highlights the DB2 9 for z/OS enhancements that directly impact DB2 application developers. Examples of areas this presentation will cover include:

- New data types and functions
- New SQL statements like INTERSECT, EXCEPT, MERGE, and TRUNCATE
- The ability to SELECT FROM and UPDATE, DELETE, or MERGE statement
- Improvements to existing SQL
- A (very) brief overview of DB2 9 XML capability
- And more…

NEON Enterprise Software, Inc.

DB2 9 for z/OS

General Availability: March 2007

DB2 9 – where's the "V"?

V8 was large… *sometimes painful*

DB2 9 nowhere near as intimidating

But it offers some nice new development "things" and "stuff"

NEON
Enterprise Software, Inc.

SQL Compatibility: DB2 z V8 / DB2 LUW V8.2

z

Multi-row INSERT, FETCH & multi-row cursor UPDATE, Dynamic Scrollable Cursors, GET DIAGNOSTICS, Enhanced UNICODE for SQL, join across encoding schemes, IS NOT DISTINCT FROM, Session variables, range partitioning

common

Inner and Outer Joins, Table Expressions, Subqueries, GROUP BY, Complex Correlation, Global Temporary Tables, CASE, 100+ Built-in Functions including SQL/XML, Limited Fetch, Insensitive Scroll Cursors, UNION Everywhere, MIN/MAX Single Index Support, Self Referencing Updates with Subqueries, Sort Avoidance for ORDER BY, and Row Expressions, 2M Statement Length, GROUP BY Expression, Sequences, Scalar Fullselect, Materialized Query Tables, Common Table Expressions, Recursive SQL, CURRENT PACKAGE PATH, VOLATILE Tables, Star Join Sparse Index, Qualified Column names, Multiple DISTINCT clauses, ON COMMIT DROP, Transparent ROWID Column, Call from trigger, statement isolation, FOR READ ONLY KEEP UPDATE LOCKS, SET CURRENT SCHEMA, Client special registers, long SQL object names, SELECT from INSERT

LUW

Updateable UNION in Views, ORDER BY/FETCH FIRST in subselects & table expressions, GROUPING SETS, ROLLUP, CUBE, INSTEAD OF TRIGGER, EXCEPT, INTERSECT, 16 Built-in Functions, MERGE, Native SQL Procedure Language, SET CURRENT ISOLATION, BIGINT data type, file reference variables, SELECT FROM UPDATE or DELETE, multi-site join, MDC

NEON Enterprise Software, Inc.

SQL Compatibility: DB2 9 – z vs. LUW

z

Multi-row INSERT, FETCH & multi-row cursor UPDATE, Dynamic Scrollable Cursors, GET DIAGNOSTICS, Enhanced UNICODE for SQL, join across encoding schemes, IS NOT DISTINCT FROM, Session variables, **TRUNCATE, DECIMAL FLOAT, VARBINARY, optimistic locking, FETCH CONTINUE, ROLE, MERGE, SELECT from MERGE**

c o m m o n

Inner and Outer Joins, Table Expressions, Subqueries, GROUP BY, Complex Correlation, Global Temporary Tables, CASE, 100+ Built-in Functions including SQL/XML, Limited Fetch, Insensitive Scroll Cursors, UNION Everywhere, MIN/MAX Single Index Support, Self Referencing Updates with Subqueries, Sort Avoidance for ORDER BY, and Row Expressions, 2M Statement Length, GROUP BY Expression, Sequences, Scalar Fullselect, Materialized Query Tables, Common Table Expressions, Recursive SQL, CURRENT PACKAGE PATH, VOLATILE Tables, Star Join Sparse Index, Qualified Column names, Multiple DISTINCT clauses, ON COMMIT DROP, Transparent ROWID Column, Call from trigger, statement isolation, FOR READ ONLY KEEP UPDATE LOCKS, SET CURRENT SCHEMA, Client special registers, long SQL object names, SELECT from INSERT, **UPDATE or DELETE, INSTEAD OF TRIGGER, Native SQL Procedure Language, BIGINT, file reference variables, XML, FETCH FIRST & ORDER BY in subselect and fullselect, caseless comparisons, INTERSECT, EXCEPT, not logged tables, OmniFind, Spatial, range partitioning, compression**

L U W

Updateable UNION in Views, GROUPING SETS, ROLLUP, CUBE, 16 Built-in Functions, SET CURRENT ISOLATION, multi-site join, MERGE, MDC, **XQuery**

NEON Enterprise Software, Inc.

SQL Compatibility: DB2 9 – z vs. DB2 9.5 LUW

z

Multi-row INSERT, FETCH & multi-row cursor UPDATE, Dynamic Scrollable Cursors, GET DIAGNOSTICS, Enhanced UNICODE for SQL, join across encoding schemes, IS NOT DISTINCT FROM, TRUNCATE, VARBINARY, FETCH CONTINUE, MERGE, SELECT from MERGE, index compression

common

Inner and Outer Joins, Table Expressions, Subqueries, GROUP BY, Complex Correlation, Global Temporary Tables, CASE, 100+ Built-in Functions including SQL/XML, Limited Fetch, Insensitive Scroll Cursors, UNION Everywhere, MIN/MAX Single Index, Self Referencing Updates with Subqueries, Sort Avoidance for ORDER BY, and Row Expressions, 2M Statement Length, GROUP BY Expression, Sequences, Scalar Fullselect, Materialized Query Tables, Common Table Expressions, Recursive SQL, CURRENT PACKAGE PATH, VOLATILE Tables, Star Join Sparse Index, Qualified Column names, Multiple DISTINCT clauses, ON COMMIT DROP, Transparent ROWID Column, Call from trigger, statement isolation, FOR READ ONLY KEEP UPDATE LOCKS, SET CURRENT SCHEMA, Client special registers, long SQL object names, SELECT from INSERT, UPDATE or DELETE, INSTEAD OF TRIGGER, Native SQL Procedure Language, BIGINT, file reference variables, XML, FETCH FIRST & ORDER BY in subselect & fullselect, caseless comparisons, INTERSECT, EXCEPT, not logged tables, OmniFind, spatial, range partitions, data compression, **session variables, DECIMAL FLOAT, optimistic locking, ROLE**

luw

Updateable UNION in Views, GROUPING SETS, ROLLUP, CUBE, **more** Built-in Functions, SET CURRENT ISOLATION, multi-site join, MERGE, MDC, XQuery, **XML enhancements, array data type, global variables, Oracle syntax**

NEON Enterprise Software, Inc.

DB2 9 New SQL Stuff

- MERGE
- TRUNCATE
- SELECT FROM UPDATE, DELETE, MERGE
- INTERSECT
- EXCEPT
- INSTEAD OF TRIGGER
- Native SQL Procedure Language
- REOPT Enhancements
- Histogram statistics
- PLAN Management
- LOB Improvements
- FETCH FIRST and ORDER BY in subselect and fullselect
- Index on expressions
- New built-in functions
- Skip Locked Data
- Optimistic Locking
- New Data Types
- XML

NEON Enterprise Software, Inc.

New SQL Statement: MERGE

Have you ever had a program requirement to accept input, either from a file or online, and take the following action(s):

- If the data is not in the database, INSERT it
- If the data is in the database, UPDATE the existing row with any new values provided

A common requirement is to present data using a spreadsheet metaphor, where rows can be inserted or modified on the screen.

MERGE

MERGE combines UPDATE and INSERT into a single SQL statement

- When source row matches target, the target row is updated
- When source row does not match, the source row is inserted into the target

MERGE Syntax

MERGE INTO *table_name*

USING *table_name*

ON (*condition*)

WHEN MATCHED THEN

UPDATE SET *column1* = *value1* [, *column2* = *value2* …]

WHEN NOT MATCHED

THEN INSERT *column1* [, *column2* …] **VALUES** (*value1* [, *value2* …]) ;

NEON Enterprise Software, Inc.

MERGE Example

Target Table

Source Table

```
MERGE INTO CUST C
  USING VALUES
    ((:CUSTNO, :CUSTNAME, :CUSTDESC)
      FOR :HV_NROWS ROWS) AS NEW (CUSTNO, NAME, DESC)
  ON (C.CUSTNO = NEW.CUSTNO)
  WHEN MATCHED THEN UPDATE
    SET (C.NAME, C.DESC) = (NEW.NAME, NEW.DESC)
  WHEN NOT MATCHED THEN INSERT (CUSTNO, NAME, DESC)
    VALUES (NEW.CUSTNO, NEW.NAME, NEW.DESC)
  NOT ATOMIC CONTINUE ON SQL EXCEPTION;
```


NEON Enterprise Software, Inc.

What Happens if we MERGE?

No	Name	Desc
10	Joe	Clerk
20	Sally	Admin
35	George	Boss
40	Bill	Foreman
52	Stacy	CEO

Target Table

No	Name	Desc
20	Sally	Manager
30	Vincent	Driver
52	Stacey	CEO

Source Table

MERGE Results

	No	Name	Desc
	10	Joe	Clerk
Edited Row →	20	Sally	**Manager**
New Row →	**30**	**Vincent**	**Driver**
	35	George	Boss
	40	Bill	Foreman
Edited Row →	52	**Stacey**	CEO

Target Table

NEON Enterprise Software, Inc.

New SQL Statement: TRUNCATE

The TRUNCATE statement is simply a quick way to DELETE *all* of the data from a table.

- The table can be in any type of table space and it can be either a base table or a declared global temporary table.
- If the table contains LOB or XML columns, the corresponding table spaces and indexes are also truncated.
- DELETE triggers will *not* be fired.

TRUNCATE Example

```
TRUNCATE TABLE EXAMPLE_TABLE

REUSE STORAGE

IGNORE DELETE TRIGGERS

IMMEDIATE;
```

REUSE STORAGE tells DB2 to empty allocated storage but keep it allocated. The alternate (default) is DROP STORAGE, which tells DB2 to release the storage that is allocated and to make it available for use (for that table or any other table in the table space).

IGNORE DELETE TRIGGERS tells DB2 to not fire any DELETE triggers. Alternately, you can specify RESTRICT WHEN DELETE TRIGGERS, which will return an error if there are any delete triggers defined on the table.

The IMMEDIATE option causes the TRUNCATE to be immediately executed and it cannot be undone. If IMMEDIATE is not specified you can issue a ROLLBACK to undo the TRUNCATE.

TRUNCATE Considerations

TRUNCATE can delete all data without actually processing each physical page as long as the table is in a segmented table space or a universal table space… *unless*…

TRUNCATE will have to process each data page if:

- Table is in a simple table space
- Table is in a partitioned table space
- Table has CDC enabled, uses MLS, or VALIDPROC

New SQL: SELECT FROM DELETE, UPDATE, MERGE

What if you need to read automatically generated data? For example:

- default values (especially dates, times, and user-defined defaults)
- IDENTITY columns

In some cases, it is possible to perform actions on an inserted row before it gets saved to disk. For example, a BEFORE TRIGGER might change data before it is even recorded to disk.

- But the application program will not have any knowledge of this change that is made in the trigger.

What if the program needs to know the final column values? Previously, this was difficult and inefficient to implement.

First, a V8 Refresher: SELECT FROM INSERT

SELECT FROM INSERT, introduced in DB2 V8 allows you to both insert the row and retrieve the values of the columns with a single SQL statement. It performs very well because it performs both the INSERT and the SELECT as a single operation.

Consider the following example:

```
SELECT COL5 INTO :C5-HV
   FROM FINAL TABLE
   (INSERT (COL1, COL2, COL5, COL7) INTO SAMPLE_TABLE
    VALUES('JONES', 'CHARLES', CURRENT DATE, 'HOURLY')
   );
```

V9 Adds DELETE, UPDATE, and MERGE Support

DB2 V9 allows the FROM clause of a SELECT statement† to contain a searched UPDATE, a searched DELETE, or a MERGE statement.

This allows the user, or a program, to know which values were updated or deleted.

† a SELECT statement that is a subselect; or in a SELECT INTO statement

An Example: SELECT FROM UPDATE

```
SELECT SUM(SALARY) INTO :SAL-HV
FROM FINAL_TABLE
 (UPDATE EMP
   SET SALARY = SALARY * 1.02
   WHERE WORKDEPT = 'A01');
```

Prior to the capability you would have had to run the UPDATE statement, and then only after it finishes, you would run the SELECT to add up the new salary values.

Now, instead of multiple statements requiring multiple passes through the data, you can consolidate it into one.

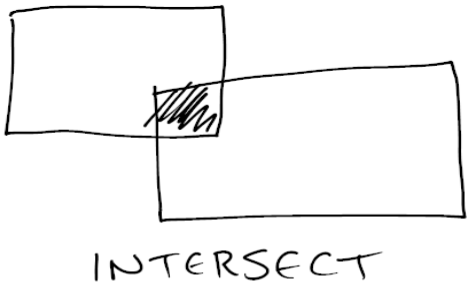
INCLUDE: SELECT FROM MERGE

And because you might want to know which rows were updated and which were inserted, you can INCLUDE a special column on MERGE:

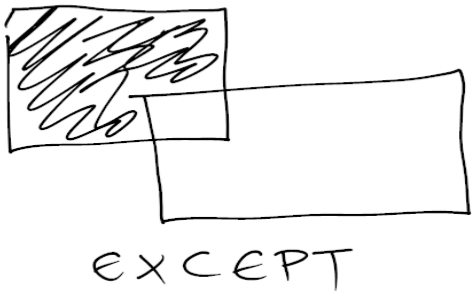
```
SELECT CUSTNO, STATUS FROM FINAL TABLE
(MERGE INTO CUST C  INCLUDE (STATUS CHAR(3) )
USING VALUES ((:CUSTNO, :CUSTNAME, :CUSTDESC) FOR :HV_NROWS ROWS)
                    AS NEW (CUSTNO, NAME, DESC)
ON (C.CUSTNO = NEW.CUSTNO)
WHEN MATCHED THEN
   UPDATE SET (C.NAME = NEW.NAME, C.DESC = NEW.DESC, STATUS = 'UPD')
WHEN NOT MATCHED THEN
    INSERT (CUSTNO, NAME, DESC) VALUES (NEW.CUSTNO, NEW.NAME, NEW.DESC, 'INS')
NOT ATOMIC CONTINUE ON SQL EXCEPTION;
```


NEON Enterprise Software, Inc.

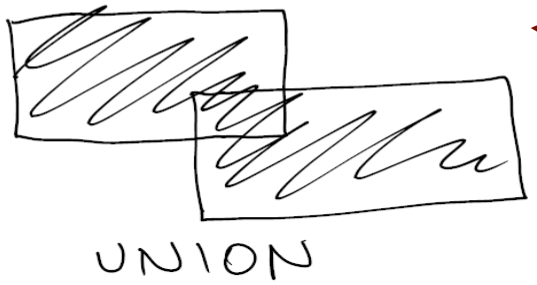
Set Operations: INTERSECT, EXCEPT, and UNION



INTERSECT is used to match result sets between two tables. If the data is the same in both results sets it passes through. When INTERSECT ALL is specified, the result consists of all rows that are in both result sets. If INTERSECT is specified without the ALL option, the duplicates will be removed from the results.



EXCEPT, on the other hand, combines non-matching rows from two result tables. Some other DBMS implementations refer to this as the MINUS operation. When EXCEPT ALL is specified, the result consists of all rows from the first result table that do not have a corresponding row in the second and any duplicate rows are kept. If EXCEPT is specified without the ALL option, duplicates are eliminated.



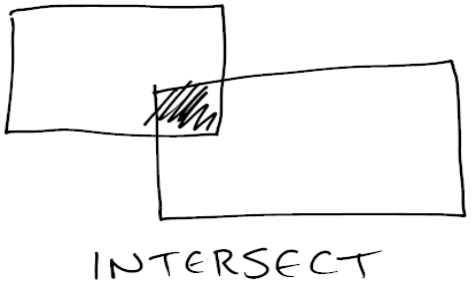
Both of these new set operations work similarly to UNION, which has existed since the beginning days of DB2. UNION gives all rows in both tables regardless of which they originated from. UNION ALL keeps duplicates whereas UNION removes duplicates.

NEON
Enterprise Software, Inc.

INTERSECT Example

For example, the following SQL will show all customers in the USA who are also employees (with no duplicates):

```
SELECT last_name, first_name, cust_num
FROM CUST
WHERE country = 'USA'
INTERSECT
SELECT last_name, first_name, emp_num
FROM EMP
WHERE country = 'USA';
```

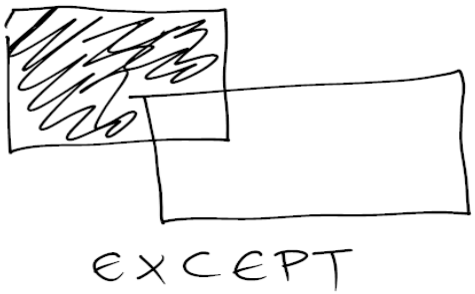


NEON
Enterprise Software, Inc.

EXCEPT Example

The following SQL will return only those items sold in March that were not also sold in April:

```
SELECT item FROM MARCH_SALES
EXCEPT
SELECT item FROM APRIL_SALES;
```



NEON
Enterprise Software, Inc.

INTERSECT and EXCEPT Considerations

The SELECT-lists must be UNION compatible. That is, they must have the same number of columns and the data type and length for each respective column must be compatible.

Cannot be used with CLOB, BLOB, DBCLOB, or XML data types.

More on SQL set operations Wikipedia at:

- http://en.wikipedia.org/wiki/Union_(SQL)

New Type of Trigger: INSTEAD OF

INSTEAD OF triggers can only be defined on VIEWs.

INSTEAD OF triggers enable views that would not otherwise be updatable to support updates.

Typically, a view that consists of multiple base tables cannot be updated.

With an INSTEAD OF trigger you can code logic to direct inserts, updates and deletes to the appropriate underlying tables that comprise the view.

INSTEAD OF Trigger Example (The View)

Let's take a look at an example to better understand the INSTEAD OF trigger. First, we create a view that joins the EMP and DEPT tables:

```
CREATE VIEW EMP_DEPT
   (EMPNO, FIRSTNME, MIDINIT, LASTNAME, PHONENO,
    HIREDATE, DEPTNAME) AS
SELECT EMPNO, FIRSTNME, MIDINIT, LASTNAME, PHONENO,
          HIREDATE, DEPTNAME
          FROM EMP, DEPT
WHERE EMP.WORKDEPT = DEPT.DEPTNO;
```

INSTEAD OF Trigger Example (Triggers #1)

Since this view is a join, it is not updateable. We can now remedy this by coding up some INSTEAD OF triggers. First, we'll take care of **INSERTs**:

An insert against the view would not be inserting a new department, so we will be inserting data into the EMP table. If the department does not exist, we'll raise an error.

```
CREATE TRIGGER E_D_ISRT INSTEAD OF INSERT ON EMP_DEPT
REFERENCING NEW AS NEWEMP FOR EACH ROW
INSERT INTO EMPLOYEE (EMPNO, FIRSTNME, MIDINIT, LASTNAME,
                      WORKDEPT, PHONENO, HIREDATE)
    VALUES(EMPNO, FIRSTNME, MIDINIT, LASTNAME,
           COALESCE((SELECT DEPTNO FROM DEPT AS D
                     WHERE D.DEPTNAME = NEWEMP.DEPTNAME),
                    RAISE_ERROR('70001', 'Unknown dept name')),
           PHONENO, HIREDATE);
```

NEON Enterprise Software, Inc.

INSTEAD OF Trigger Example (Triggers #2)

Next we'll consider updates:

```
CREATE TRIGGER E_D_UPD INSTEAD OF UPDATE ON EMP_DEPT
REFERENCING NEW AS NEWEMP OLD AS OLDEMP
  FOR EACH ROW
BEGIN ATOMIC
 VALUES(CASE WHEN NEWEMP.EMPNO = OLDEMP.EMPNO THEN 0
             ELSE RAISE_ERROR('70002', 'Must not change EMPNO') END);
UPDATE EMP AS E
   SET (FIRSTNME, MIDINIT, LASTNAME, WORKDEPT, PHONENO, HIREDATE)
    = (NEWEMP.FIRSTNME, NEWEMP.MIDINIT, NEWEMP.LASTNAME,
           COALESCE((SELECT DEPTNO FROM DEPT AS D
                     WHERE D.DEPTNAME = NEWEMP.DEPTNAME),
                    RAISE_ERROR ('70001', 'Unknown dept name')),
          NEWEMP.PHONENO, NEWEMP.HIREDATE)
 WHERE NEWEMP.EMPNO = E.EMPNO;END
```

An update against the view would be updating employee info, so we will be updating data in the EMP table. If the department does not exist, we'll raise an error.

NEON Enterprise Software, Inc.

INSTEAD OF Trigger Example (Triggers #3)

Finally we take care of deletions:

```
CREATE TRIGGER E_D_DEL INSTEAD OF DELETE ON EMP_DEPT
REFERENCING OLD AS OLDEMP FOR EACH ROW
DELETE FROM EMP AS E WHERE E.EMPNO = OLDEMP.EMPNO;
```

Again, deleting from this view would impact employee data, not department data, so we will code the trigger to delete the appropriate rows from the EMP table.

INSTEAD OF Trigger Considerations - 1

The view must exist at the current server and none of the following are permitted for a view to have an INSTEAD OF trigger:

- the WITH CASCADED CHECK option
- a view on which a symmetric view has been defined
- a view that references data encoded with different encoding schemes or CCSID values
- a view with a ROWID, LOB, or XML column (or a distinct type that is defined as one of these types)
- a view with a column based on an underlying column defined as an identity column, security label column, or a row change timestamp column
- a view with a column that is defined (directly or indirectly) as an expression
- a view with a column that is based on a column of a result table that involves a set operator
- a view with any columns that have field procedures
- a view where all of the underlying base tables are DB2 Catalog tables or created global temporary tables
- a view that has other views dependent on it

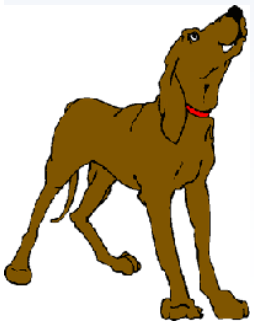
INSTEAD OF Trigger Considerations - 2

Using an INSTEAD OF trigger, each requested modification operation made against the view is replaced by the trigger logic. The trigger performs the insert, update, or delete on behalf of the view. No application changes are required because the code is in the trigger which resides in the database.

Only **one** INSTEAD OF trigger is allowed for each type of operation on a given subject view.

If you want to read more about INSTEAD OF triggers, check out this extensive article (*albeit for DB2 LUW*) available on the IBM Developer Works web site:

http://www.ibm.com/developerworks/db2/library/techarticle/0210rielau/0210rielau.html



NEON Enterprise Software, Inc.

Native SQL Procedure Language Improvements

Native SQL stored procedures contain only SQL procedure language (PL) statements.

In V8:

- Native SQL stored procedures are converted to C, which must be compiled (and obviously, requires a C compiler), and stored in the DB2 Catalog.

In DB2 9:

- Native SQL stored procedures are converted to a native representation before they are stored in the DB2 catalog. Native SQL stored procedures do not run under WLM, but in the DBM1 address space. This improves performance.
- Native SQL stored procedures that are run through DDF are eligible to run on zIIP processors.
- Stored procedures can be versioned, making it easer to develop & test them.
- Nested compound SQL statements.

You have to drop and re-create native stored procedures after moving to DB2 9 to take advantage of these changes.

Reoptimization

You can gain additional optimization for dynamic SQL using the REOPT parameter of the BIND command.

REOPT specifies whether to have DB2 determine an access path at run time by using the values of host variables, parameter markers, and special registers.

As of DB2 9, there are four options from which to choose when specifying REOPT.

NEON Enterprise Software, Inc.

REOPT Enhancements

NOREOPT(VARS) can be specified as a synonym of REOPT(NONE).

REOPT (NONE) –PREPARE determines the access path and no reoptimization is performed. The bound statement can be moved to the dynamic statement cache (DSC), if the cache is being used.

DB2 V8

REOPT (ONCE) – PREPARE determines an initial access path before the host variable values are available. When the statement is first executed and the host variable values are known, the statement is reoptimized one time. The hope is that the one-time reoptimization will provide a better access path than the initial PREPARE. The statement can be placed in the dynamic statement cache and reused multiple times.

REOPT (ALWAYS) – The SQL statement is re-optimized each time it is executed, always using the latest host variable values.

REOPT(VARS) can be specified as a synonym of REOPT(ALWAYS).

REOPT (AUTO) – Leave it up to DB2 (autonomic?). If changes in the filter factors for the statement predicates warrant, DB2 can re-prepare the statement. The newly prepared statement would be executed and would replace the prepared statement currently in the Global Dynamic Statement cache.

DB2 9

REOPT Applicability: Dynamic vs. Static SQL

REOPT Parameter	Dynamic SQL	Static SQL
NONE	YES	YES
ALWAYS	YES	YES
ONCE	YES	NO
AUTO	YES	NO

Consider binding static programs with REOPT(ALWAYS) when the values for your program's host variables or special registers are volatile and make a significant difference for access paths.

ONCE and AUTO are not valid for static SQL because they work with the dynamic statement cache, which does not apply to static SQL.

NEON Enterprise Software, Inc.

PLANMGMT: Plan Stability

Plan stability, which works on packages only, allows you to keep backups versions of your program's access paths.

Why? Because sometimes, after rebinding your program, performance degrades. With plan stability you can fall back to a previous package.

NEON
Enterprise Software, Inc.

PLANMGMT BIND Options

Previous and active copies of package.

PLANMGMT(OFF) - No change to existing behavior. A package continues to have one active copy.

PLANMGMT(BASIC) - A package has one active copy. One additional prior copy (PREVIOUS) is preserved.

Original, previous and active copies of package.

PLANMGMT(EXTENDED) - A package has one active copy, and two additional prior copies (PREVIOUS and ORIGINAL) are preserved.

NEON Enterprise Software, Inc.

The Ol' Switcheroo

SWITCH (PREVIOUS) - changes the current and previous packages:

- The existing current package takes the place of the previous package.
- The existing previous package takes the place of the current package.

Only if you bound using PLANMGMT EXTENDED (refer to previous slide).

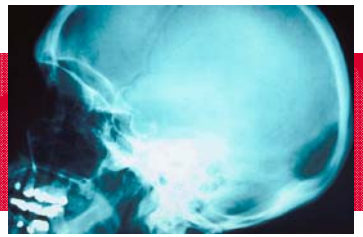
SWITCH (ORIGINAL) - clones the original copy to take the place of the current copy:

- The existing current copy replaces the previous copy.
- The existing previous copy is discarded.

Caution: PLANMGMT doubles (or triples) storage requirements for package skeletons in SPT02

NEON Enterprise Software, Inc.

LOB Improvements #1



LOB File Reference Variable

- A host variable defined in a host language to contain the file name that directs file input and output for a large object (LOB).
- Using file reference variables, large LOB values can be inserted from a file or selected into a file rather than a host variable.
- This means that your application program does not need to acquire storage to contain the LOB value.
- File reference variables also enable you to move LOB values from the DBMS to a client application or from a client application to a database server without going through the working storage of the client application.

NEON Enterprise Software, Inc.

LOB Improvements #2 - FETCH CONTINUE

Prior to Version 9, there were two methods you could deploy in your programs to fetch LOB data:

- Fetching data into a pre-allocated buffer
  - can cause virtual storage constraint problems, especially for larger LOBs)
- Using a LOB locator to retrieve a handle on the data.
  - using LOB locators that commit infrequently or do not explicitly free the locators can use considerable amounts of DB2 resources

NEON Enterprise Software, Inc.

LOB Improvements #2 - FETCH CONTINUE

DB2 9: FETCH CONTINUE

- Can retrieve LOB columns in multiple pieces without using a LOB locator.
- Can continue a FETCH operation to retrieve the remaining LOB data when truncation occurs.
- You will have to manage the buffers and reassemble the pieces of data in your application program.

NEON Enterprise Software, Inc.

FETCH FIRST and ORDER BY in Subselect

select-statement
- fullselect
  - subselect

First, some basic education… The SELECT statement is broken down into three sections:

The **select-statement** is the form of a query that can be directly specified in a DECLARE CURSOR statement, or prepared and then referenced in a DECLARE CURSOR statement. It is the thing most people think of when they think of SELECT in all its glory. If so desired, it can be issued interactively using SPUFI. The select-statement consists of a fullselect, and any of the following optional clauses: order-by, fetch-first, update, read-only, optimize-for, isolation and queryno.

A **fullselect** can be part of a select-statement, a CREATE VIEW statement, or an INSERT statement. This sometimes confuses folks as they try to put a FETCH FIRST n ROWS clause or an ORDER BY in a view or as part of an INSERT. A fullselect does *not* allow any of the following clauses: ORDER BY, FOR READ ONLY, FOR FETCH ONLY, FOR UPDATE OF, OPTIMIZE FOR, WITH, QUERYNO, and FETCH FIRST. A fullselect specifies a result table – and none of these afore-mentioned clauses apply.

Finally, a **subselect** is a component of the fullselect. A subselect specifies a result table derived from the result of its first FROM clause. The derivation can be described as a sequence of operations in which the result of each operation is input for the next.

Think of it this way…

In a subselect you specify the FROM to get the tables, the WHERE to get the conditions, GROUP BY to get aggregation, HAVING to get the conditions on the aggregated data, and the SELECT clause to get the actual columns.

In a fullselect you add in the UNION *(EXCEPT and INTERSECT)* to combine subselects and other fullselects.

Finally, you add on any optional order-by, fetch-first, update, read-only, optimize-for, isolation and queryno clauses to get the select-statement.

Well, until DB2 9 that is…

The Pre-9 Problem

In DB2 V8, the ORDER BY and FETCH FIRST n ROWS ONLY clauses are only allowed at the statement level as part of select-statement or a SELECT INTO statement. That is, you can specify the clauses as part of select-statement and write:

```
SELECT * FROM T ORDER BY c1 FETCH FIRST 1 ROW ONLY
```

But you cannot specify the clauses within the fullselect and write:

```
INSERT INTO T1
(SELECT * FROM T2 ORDER BY c1 FETCH FIRST 1 ROW ONLY)
```

Why is this a problem?

- Assume a large table
- You only want the first 1000 rows sorted in a particular order
- Coding a SELECT using FETCH FIRST and ORDER BY clauses does the trick, but the sort is done before the fetch (runs a large sort for no reason).
- As a work around you could code use a temp table, but that requires a lot more work.

The DB2 9 Solution

Allow **FETCH FIRST** and **ORDER BY** in a subselect, for example:

```
SELECT T1.EMPNO, T1.PROJNO
FROM DSN8910.EMPPROJACT T1
WHERE T1.EMPNO IN
  (SELECT T2.EMPNO
   FROM DSN8910.EMP T2
   ORDER BY SALARY DESC
   FETCH FIRST 3 ROWS ONLY)
ORDER BY T1.PROJNO;
```

Enclosed in parentheses.

Enclosed in parentheses.

The subselect **MUST** be enclosed in parentheses and the ORDER BY or FETCH FIRST cannot be:

- in the outermost fullselect of a view
- in a materialized query table

NEON Enterprise Software, Inc.

Index on Expressions

DB2 9 extends the CREATE INDEX statement to index expressions instead of just columns.

What is an expression? It can be as simple as a column reference, or it can be a built-in function invocation or even a more general expression, with certain restrictions.

NEON Enterprise Software, Inc.

Index on Expressions - Restrictions

Each expression must contain as least one reference to a column of the table named in the ON clause of the index.

Referenced columns cannot be LOB, XML, or DECFLOAT data types nor can they be a distinct type that is based on one of those data types.

Referenced columns cannot include any FIELDPROCs nor can they include a SECURITY LABEL.

And the expression **cannot** include any of the following:

- Subquery
- Non-deterministic function
- User-defined function
- Host variable
- Special register
- OLAP specification
- Aggregate function
- Function that has an external action
- Sequence reference
- Parameter marker
- CASE expression

Index on Expressions - Examples

```
CREATE INDEX XUPLN
  ON EMP
  (UPPER(LAST_NAME))
  USING STOGROUP DSN8G910
        PRIQTY 360 SECQTY 36
        ERASE NO     COPY YES;

CREATE INDEX XTOTCOMP
    ON EMP   (SALARY + COMM + BONUS) . . .

CREATE UNIQUE INDEX XLNFN
    ON EMP  (SUBSTR(FIRSTNAME,1,1) CONCAT '. ' CONCAT LASTNAME) . . .
```


NEON Enterprise Software, Inc.

New Built-in Functions – Char/String Functions

ASCII_CHR - returns the character that has the ASCII code value that is specified by the argument.

ASCII_STR - returns a string, in the system ASCII CCSID that is an ASCII version of the string.

EBCDIC_CHR / **EBCDIC_STR** - I bet you can guess what they do.

UNICODE - returns the Unicode UTF-16 code value of the leftmost character of the argument as an integer.

UNICODE_STR - returns a string in Unicode UTF-8 or UTF-16 (depending on the specified parameter) representing a Unicode encoding of the input string.

NEON Enterprise Software, Inc.

New Built-in Functions – String manipulation

LPAD - returns a string that is padded on the left, with blanks (or a specific character). The LPAD function treats leading or trailing blanks as significant.

RPAD - does the same but on the right. So, in the following example the last name is left padded with periods, then right padded with blanks.

```
SELECT LPAD(LASTNAME, 30, '.' ) AS LAST,
       RPAD(FIRSTNME, 30) AS FIRST
FROM   DSN910.EMP;
```

OVERLAY - returns a string with portions of it overlaid by a specified value. You provide the string, a substring to be overlaid, and its starting point and length, and DB2 does the rest. For example:

```
SELECT CHAR(OVERLAY('PLATELET','CEMEN',4,4,OCTETS),9),
       CHAR(OVERLAY('INSERTING','IS',4,2,OCTETS),10),
FROM   SYSIBM.SYSDUMMY1;
```

Returns: PLACEMENT

Returns: INSISTING

NEON Enterprise Software, Inc.

New Built-in Functions – "Similar Sounding"

SOUNDEX - returns a 4 character code that represents the sound of the words in the argument. The result can be used to compare with the sound of other strings. The data type of the result is CHAR(4). For example, this query would return not only employees with a last name of "Smith," but also anything that sounds like Smith, such as Smythe:

```
SELECT  LASTNAME
FROM    DSN910.EMP
WHERE   SOUNDEX(LASTNAME) = SOUNDEX('Smith');
```

DIFFERENCE - returns a value from 0 to 4 where the number represents the difference between the sounds of two strings based on applying the SOUNDEX function to the strings. The higher the value, the closer the two strings are to sounding alike. Consider this example which returns the values 4, C523, and C523:

```
SELECT DIFFERENCE('CONSTRAINT', 'CONSTANT'),
       SOUNDEX('CONSTRAINT'),
       SOUNDEX('CONSTANT')
FROM   SYSIBM.SYSDUMMY1;
```

Since the two strings return the same SOUNDEX value, the difference is 4 (the highest value possible).

NEON Enterprise Software, Inc.

New Built-in Functions – Date/Time

EXTRACT - returns a portion of a date or timestamp. You can use EXTRACT to slice up a date/time value into its component pieces. For example:

```
SELECT BIRTHDATE,
     EXTRACT (DAY FROM BIRTHDATE) AS DAY,
     EXTRACT (MONTH FROM BIRTHDATE) AS MONTH,
     EXTRACT (YEAR FROM BIRTHDATE) AS YEAR
FROM   DSN8910.EMP;
```

MONTHS_BETWEEN - returns an estimate of the number of months between two expressions. The result is calculated based on a 31 day month. For example, the result of this query would be 1.096774193548387:

```
SELECT MONTHS_BETWEEN ('2007-02-20','2007-01-17')
AS     MONTHS_BETWEEN
FROM   SYSIBM.SYSDUMMY1;
```


NEON Enterprise Software, Inc.

New Built-in Functions - Timestamp

TIMESTAMPADD - adds an interval to a timestamp.

TIMESTAMPDIFF - subtracts two timestamps and returns an interval.

TIMESTAMP_FORMAT – changes the display format for a timestamp value. Valid formats that can be specified are:

- 'YYYY-MM-DD'
- 'YYYY-MM-DD-HH24-MI-SS'
- 'YYYY-MM-DD-HH24-MI-SS-NNNNNN'

NEON Enterprise Software, Inc.

Other New Built-in Functions

RID - returns the RID of a row

CORRELATION - returns the coefficient of correlation of a set of number pairs

COVARIANCE / COVARIANCE_SAMP - return the (population) covariance of a set of number pairs

And, of course, we also get scalar functions to support the new data types in DB2 9 (the new data types are discussed later in this presentation).

OLAP Functions: RANK

This query will rank employees who have total compensation greater than $30,000, but order the results by last name. This allows you to rank data differently than the order in which it is presented.

```
SELECT EMPNO, LASTNAME, FIRSTNAME,
       SALARY+BONUS+COMM AS TOTAL_COMP,
       RANK()
       OVER(ORDER BY SALARY+BONUS+COMM DESC)
          AS RANK_COMP
FROM EMP
WHERE SALARY+BONUS+COMM > 30000
ORDER BY LASTNAME;
```

OLAP Functions: RANK

OK, what did that query do?

EMPNO	LASTNAME	FIRSTNAME	SALARY	BONUS	COMM
100	MULLINS	CRAIG	500000	100000	400000
200	SHAW	DONNA	25000	10000	0
300	ABBOTT	BRANDY	700000	300000	0
400	WISER	BUD	10000	0	0
500	GREEN	RACHEL	40000	2000	5000

300	ABBOTT	BRANDY	1000000	1
500	GREEN	RACHEL	47000	3
100	MULLINS	CRAIG	1000000	1
200	SHAW	DONNA	35000	4
400	WISER	BUD	10000	5

RANK of the total compensation

NEON Enterprise Software, Inc.

OLAP Functions: DENSE_RANK

Both ABBOTT and MULLINS earn the most, but the amount is the same, so they share the number one ranking. With a dense rank, the next rank value is 2, and not 3.

EMPNO	LASTNAME	FIRSTNAME	SALARY	BONUS	COMM
100	MULLINS	CRAIG	500000	100000	400000
200	SHAW	DONNA	25000	10000	0
300	ABBOTT	BRANDY	700000	300000	0
400	WISER	BUD	10000	0	0
500	GREEN	RACHEL	40000	2000	5000

300	ABBOTT	BRANDY	1000000	1
500	GREEN	RACHEL	47000	2
100	MULLINS	CRAIG	1000000	1
200	SHAW	DONNA	35000	3
400	WISER	BUD	10000	4

When there are two at the top, next is 2 not 3 w/DENSE_RANK.

NEON Enterprise Software, Inc.

OLAP Functions – ROW_NUMBER

ROW_NUMBER - specifies that a sequential row number is computed for the row that is defined by the ordering, starting with 1 for the first row.

```
SELECT ROW_NUMBER() OVER(ORDER BY WORKDEPT,
       LASTNAME) AS NUMBER, LASTNAME, SALARY
FROM   EMP
ORDER BY WORKDEPT, LASTNAME;
```


NEON Enterprise Software, Inc.

Skipping Locked Data

In DB2 9 it is possible to set up your SQL to skip over pages or rows that are locked.

The program will only "see" unlocked, committed data.

Why would you want to do this? Perhaps to improve application performance and availability. If you don't have to wait for locks to be released then "things" should run faster, right?

But it comes at the cost of not accessing the locked data at all. Yes, this means your results will be inconsistent. You should only utilize this clause when your program can tolerate skipping over some data and NEVER when results must be 100% accurate, like when balancing financial data, for example.

This is not the same as dirty reads (UR), which can "see" uncommitted locked data.

OK, How Do You Skip Locked Data?

SKIP LOCKED DATA – can be specified on:

- SELECT
- PREPARE
- searched UPDATE and DELETE
- UNLOAD utility

Can only be used with the following isolation levels:

- Cursor Stability (CS)
- Read Stability (RS)
- If any other isolation level is in use for the plan or package, the SKIP LOCKED DATA option is ignored.

NEON Enterprise Software, Inc.

For Example

UPDATE EMP
SET COMM = 500
WHERE BONUS = 0;

Assume row level locking and no COMMIT.

EMPNO	LASTNAME	FIRSTNAME	SALARY	BONUS	COMM
100	MULLINS	CRAIG	500000	100000	400000
200	SHAW	DONNA	25000	0	0
300	ABBOTT	RANDY	700000	300000	0
400	WISER	BUD	10000	2000	0
500	GREEN	RACHEL	40000	0	5000

Locked

Locked

```
SELECT COUNT (*)
FROM    EMP
WHERE COMM = 0
SKIP LOCKED DATA;
```

→ 2

```
SELECT COUNT (*)
FROM    EMP
SKIP LOCKED DATA;
```

→ 3

NEON Enterprise Software, Inc.

Optimistic Locking

What is optimistic locking?

- We are optimists and think that usually we will be the only ones with interest in the data.
- In other words, when optimistic locking is implemented you are assuming that most of the time there will be no other programs that are interested in the page of data that you are planning to modify.
- Optimistic locking decreases the duration of locks and can improve performance and availability, helping to eliminate -911 and -913 locking issues.

NEON Enterprise Software, Inc.

Optimistic Locking and ROW CHANGE TIMESTAMP

For programs that use updateable static scrollable cursors, DB2 can use optimistic locking as long as the plan/package is bound ISOLATION(CS).

If you have this situation, DB2 will deploy optimistic locking to reduce the duration of locks between consecutive FETCH operations and between fetch operations and subsequent positioned UPDATE or DELETE operations.

Regular Locking

FETCH row 1
LOCK row 1

FETCH row 2
UNLOCK row 1
LOCK row 2

UPDATE row 2

Optimistic Locking

FETCH row 1
LOCK row 1
UNLOCK row 1

FETCH row 2
LOCK row 2
UNLOCK row 2

UPDATE row 2
LOCK row 2
re-evaluate and compare
UPDATE

NEON
Enterprise Software, Inc.

ROW CHANGE TIMESTAMP

The ROW CHANGE TIMESTAMP can be used to find out when table rows were modified. For example:

```
CREATE TABLE CUSTOMER
(CUSTNO          CHAR(8)   NOT NULL,
 CUST_INFOCHANGE  NOT NULL GENERATED ALWAYS
                  FOR EACH ROW ON UPDATE
                  AS ROW CHANGE TIMESTAMP,
 CUST_NAME        VARCHAR(50),
 CUST_ADDRESS     VARCHAR(100),
 CUST_CITY        CHAR(20),
 CUST_STATE       CHAR(2),
 CUST_ZIP         CHAR(9),
 CUST_PHONE       CHAR(10),
 PRIMARY KEY (CUSTNO))
```

Now that the table is defined with the ROW CHANGE TIMESTAMP we can use it in our programs and queries to determine when the row was last changed.

NEON Enterprise Software, Inc.

Using the ROW CHANGE TIMESTAMP

To find all of the customer rows that were changed in the past week (ie. the last 7 days) we could run the following query:

```
SELECT CUSTNO, CUST_NAME
FROM   CUSTOMER
WHERE  ROW CHANGE TIMESTAMP FOR CUSTOMER <=
       CURRENT TIMESTAMP
AND    ROW CHANGE TIMESTAMP FOR CUSTOMER >=
       CURRENT TIMESTAMP - 7 DAYS;
```

ALTER and ROW CHANGE TIMESTAMP

But what would happen if you issued a statement like on the previous slide, but against a table that was **altered** to include a ROW CHANGE TIMESTAMP, not created initially with one?

DB2 will use the time the **page** was last modified until your REORG. So the results will not be exactly correct because it would return all the rows on each page that qualifies (because at least one row on the page changed).

Clean up the advisory REORG pending as soon as possible after adding the ROW CHANGE TIMESTAMP.

NEON Enterprise Software, Inc.

New Data Type: BIGINT

BIGINT is an exact numeric data type capable of representing 63-bit integers. This is the third integer data type now available to DB2 and it offers the ability to store the largest range of values:

- SMALLINT values can range from -32768 to 32767
- INTEGER values can range from -2147483648 to 2147483647
- BIGINT values can range from -9223372036854775808 to 9223372036854775807

NEON Enterprise Software, Inc.

New Data Types: Binary Data

BINARY and VARBINARY data types extend current support of binary strings and are compatible with BLOB data type. They are not compatible with character string data types.

It is somewhat easy to migrate existing columns defined as CHAR FOR BIT DATA or VARCHAR FOR BIT DATA over to BINARY or VARBINARY. If there is an index defined on the column, the index is placed in RBDP.

- You cannot alter BINARY or VARBINARY data types to CHAR FOR BIT DATA or VARCHAR FOR BIT DATA.

New Data Type: DECFLOAT

DECFLOAT takes advantage of new System z9 hardware support delivering a data type that lets you use decimal floating-point numbers with greater precision than the existing FLOAT data type. The maximum precision is 34 digits and the range of a decimal floating point number is either 16 or 34 digits of precision. The range of values supported by DECFLOAT is:

- DECFLOAT(16) range: $-9.999999999999999 \times 10^{384}$ to $9.999999999999999 \times 10^{384}$
- Smallest positive DECFLOAT(16): $1.000000000000000 \times 10^{-383}$
- Largest negative DECFLOAT(16): $-1.000000000000000 \times 10^{-383}$
- DECFLOAT(34) range: $-9.999999999999999999999999999999999 \times 10^{6144}$ thru $9.999999999999999999999999999999999 \times 10^{6144}$
- Smallest positive DECFLOAT(34): $1.000000000000000000000000000000000 \times 10^{-6143}$
- Largest negative DECFLOAT(34): $-1.000000000000000000000000000000000 \times 10^{-6143}$

Additional DECFLOAT Considerations

The DECFLOAT data type is able to represent the following named special values representing “non-number numbers”:

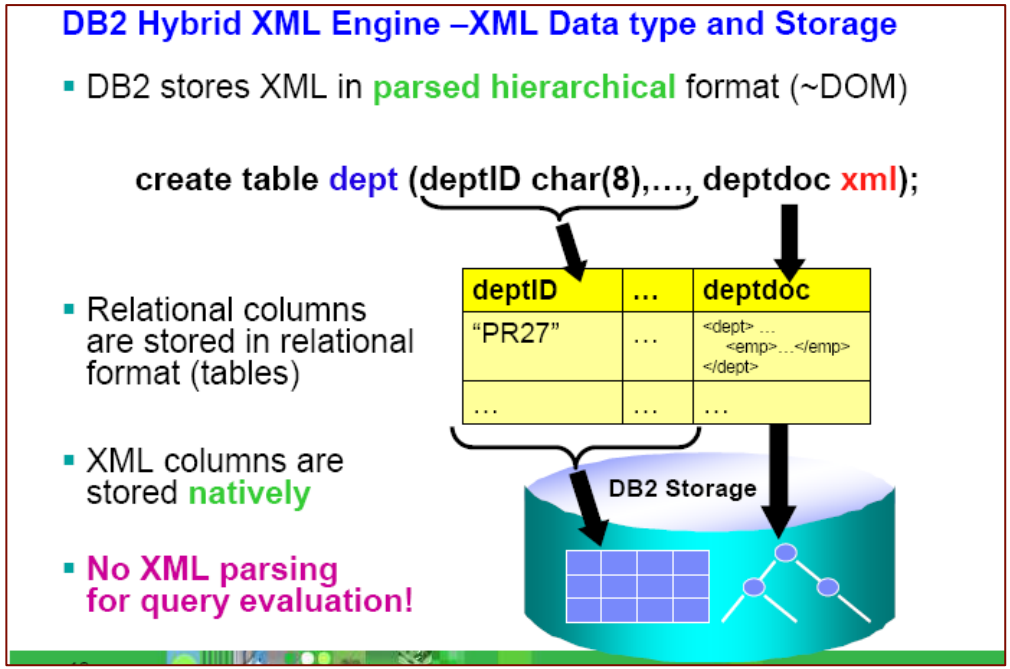
- Infinity - a value that represents a number whose magnitude is infinitely large.
- Quiet NaN - a value that represents undefined results which does not cause an invalid number condition. *NaN is not a number.*
- Signaling NaN - a value that represents undefined results which will cause an invalid number condition if used in any numerical operation.

DECFLOAT is only supported in Java, Assembler, and REXX.

pureXML

And finally, we get pureXML support to store XML as a native data type in DB2.

- That means you can specify XML as a data type for columns in your DB2 tables in DB2 9 for z/OS.



NEON Enterprise Software, Inc.

Before we go...

Plans and packages that were bound before DB2 V4 will be automatically rebound when first accessed by DB2 9, so:

- It is a good idea to rebind these yourself, ahead of time, before you try to run them in DB2 9.
- That way, you can see what the impact of the new version is on your access paths.

NEON Enterprise Software, Inc.

Summary

DB2 9 for z/OS offers a wealth of new things and stuff for DB2 application developers.

- Make sure you take the time to learn how these new features can help you build better DB2 applications!

...then maybe the work won't keep piling up like this!

NEON
Enterprise Software, Inc.

Session: G07

DB2 9: For Developers Only!

Craig S. Mullins

NEON Enterprise Software, Inc.

Craig.Mullins@neonesoft.com